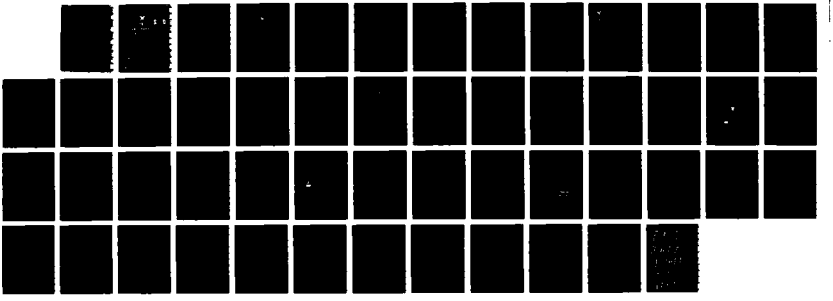
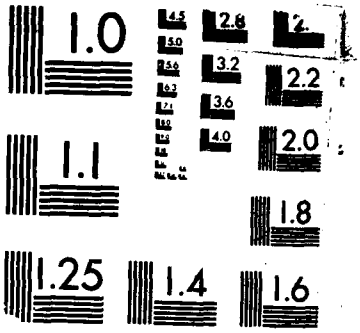
AD-A194 407 ADP (AUTOMATED DATA PROCESSING) REQUIREMENTS DEFINITION AND DOCUMENTATION THROUGH USER-DEVELOPED PROTOTYPES(U) AIR COMMAND AND STAFF COLL MAXWELL AFB AL D W MANIFEN 1/1

UNCLASSIFIED APR 88 ACSC-88-1150 F/G 12/5 NL





AIR COMMAND AND STAFF COLLEGE

STUDENT REPORT

ADP REQUIREMENTS DEFINITION AND DOCUMENTATION THROUGH USER-DEVELOPED PROTOTYPES

MAJOR DAN W. HANIFEN 88-1150

"insights into tomorrow"

DISCLAIMER

The views and conclusions expressed in this document are those of the author. They are not intended and should not be thought to represent official ideas, attitudes, or policies of any agency of the United States Government. The author has not had special access to official information or ideas and has employed only open-source material available to any writer on this subject.

This document is the property of the United States Government. It is available for distribution to the general public. A loan copy of the document may be obtained from the Air University Interlibrary Loan Service (AUL/LDEX, Maxwell AFB, Alabama, 36112-5564) or the Defense Technical Information Center. Request must include the author's name and complete title of the study.

This document may be reproduced for use in other research reports or educational pursuits contingent upon the following stipulations:

- Reproduction rights do not extend to any copyrighted material that may be contained in the research report.

- All reproduced copies must contain the following credit line: "Reprinted by permission of the Air Command and Staff College."

- All reproduced copies must contain the name(s) of the report's author(s).

- If format modification is necessary to better serve the user's needs, adjustments may be made to this report--this authorization does not extend to copyrighted information or material. The following statement must accompany the modified document: "Adapted from Air Command and Staff College Research Report (number) entitled (title) by (author)."

- This notice must be included with any reproduced or adapted portions of this document.

REPORT NUMBER 88-1150

TITLE ADP REQUIREMENTS DEFINITION AND DOCUMENTATION THROUGH USER-DEVELOPED PROTOTYPES

AUTHOR(S) MAJOR DAN W. HANIFEN, USAF

FACULTY ADVISOR MAJOR TOM HALL, ACSC/3821STUS

SPONSOR MAJOR DAN BURTON
ESD/XRS
% SEIJPO
4500 5TH AVENUE
PITTSBURGH, PA 15212

Submitted to the faculty in partial fulfillment of requirements for graduation.

AIR COMMAND AND STAFF COLLEGE
AIR UNIVERSITY
MAXWELL AFB, AL 36112-5542

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A197 427

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT STATEMENT "A" Approved for public release; Distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) 88-1150	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION ACSC/EDC 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Maxwell AFB AL 36112-5542	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
ADP REQUIREMENTS DEFINITION AND DOCUMENTATION THROUGH USER-DEVELOPED PROTOTYPES

12. PERSONAL AUTHOR(S)
Hanifen, Dan W., Major, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
	FROM _____ TO _____	1988, April	49

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The automated data processing requirements definition and documentation process does not work properly today. Users are spending more money for new or modified systems and continue to be dissatisfied with the results. Much of the dissatisfaction can be traced to seven factors that have reduced the effectiveness of the users' ability to define and communicate requirements. But, this process can be improved through user-developed prototypes supplementing the traditional ADP lifecycle. These prototypes include functioning software and hardware, a data list, a brief concept of operations, and written requirements as necessary to supplement the other documentation. User-developed prototypes offer many advantages and can vastly improve user satisfaction.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL ACSC/EDC Maxwell AFB AL 36112-5542	22b. TELEPHONE (Include Area Code) (205) 293-2867	22c. OFFICE SYMBOL

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

PREFACE

The automated data processing requirements definition and documentation process does not work properly today. Users are spending more money for new or modified systems and continue to be dissatisfied with the results. Much of the dissatisfaction can be traced to seven factors that have reduced the effectiveness of the users' ability to define and communicate requirements. But, this process can be improved through user-developed prototypes supplementing the traditional ADP lifecycle. These prototypes include functioning software and hardware, a data list, a brief concept of operations, and written requirements as necessary to supplement the other documentation. User-developed prototypes offer many advantages and can vastly improve user satisfaction. This paper records many of the author's

This paper is intended to record many of my observations over the past ~~four (4)~~ 4 years regarding the mechanics of creating, documenting and communicating requirements. It also offers a proven method to improve requirements definition through user-produced prototypes. I hope this sheds light on an otherwise cloudy subject and will be used by those in the Air Force interested in obtaining or providing better products.

I would like to extend my appreciation to Maj Edwin R. Loskill for his help producing the illustrations in this report, and to my wife, Nadine, for her continued support and typing skill.

Maj Dan W. Hanifen

Prattville, Alabama
7 Feb 1988

ABOUT THE AUTHOR

Major Dan W. Hanifen graduated in 1975 from the Air Force Academy with a Bachelors of Science in Engineering Sciences and in 1980 from the Air Force Institute of Technology with a Master of Science in Nuclear Engineering/Weapons Effects. He completed Squadron Officers School in residence in 1980 and Air Command and Staff College by correspondence in 1985. His work experience includes three years as an engineer overseeing the flight testing of reentry vehicles at Vandenberg AFB and three years supervising space systems survivability research at Kirtland AFB. Most recently, 1983-1987, Major Hanifen was the lead system analyst and primary liaison with the ADP acquisition office for SAFELM to develop (1) top-level functional requirements and systems requirements for a long term $300M ADP acquisition and (2) specifications for a $12M ADP upgrade that was successfully delivered. His duties included gathering, evaluating and documenting ADP requirements and concepts of operation/mission management. Additionally, he authored two (2) studies: a top-level operations concept definition study that has formed the basis for a prototype and the first definitive study describing operations in his organization. In his last year at SAFELM, Major Hanifen created a branch responsible for prototype development. His duties included: branch management; designing, coordinating and implementing Symbolics Lisp machine/IBM PC/AT architecture; coordinating requirements for prototype hardware and software acquisition as well as for contractor support; and, defining, assigning and monitoring five (5) prototype projects. He learned and used the Common Lisp programming language, performed site administration on the Symbolics 3600 family of Lisp computers and became familiar with Basic, C, Pascal and Fortran programming languages. He also served as a project leader for a rapid state-of-the-art prototype project using C programming language. Major Hanifen also authored the hardware architecture and software concept definition study, all operator coordination and requirements refinement, and technical supervision of four (4) contractor/programmer personnel.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

FIGURES

EXECUTIVE SUMMARY

Part of our College mission is distribution of the students' problem solving products to DoD sponsors and other interested agencies to enhance insight into contemporary, defense related issues. While the College has accepted this product as meeting academic requirements for graduation, the views and opinions expressed or implied are solely those of the author and should not be construed as carrying official sanction.

"insights into tomorrow"

REPORT NUMBER 88-1150

AUTHOR(S) MAJOR DAN W. HANIFEN, USAF

TITLE ADP REQUIREMENTS DEFINITION AND DOCUMENTATION THROUGH USER-DEVELOPED PROTOTYPES

I. Purpose: To examine the ADP requirements definition and documentation process and recommend improving it through user-developed ADP prototype systems.

II. Problem: The traditional mechanism for communicating ADP requirements is flawed. This results in poorly understood and vague requirements upon which systems are designed, developed and delivered much to the dissatisfaction of the user.

III. Current Requirements Definition Guidance: The nature of ADP requirements is briefly described and a working definition offered from AFR 700-3. Then, selected top-level Air Force Regulations (AFRs 800-14, 57-1, 55-24) and one DOD standard (DOD Std 2167) are evaluated for their contribution to understanding the complex nature of requirements or defining the roles of the participants in the definition process.

IV. Communication Model of the Requirements Definition and Documentation Process: Communication of users' requirements is ineffective for the following reasons: lack of adequate and consistent requirements definition and documentation guidance; the nature and mechanics of documenting requirements; the different responsibilities and backgrounds of the participants;

and, the environment in which the participants must work to define and document requirements. But, there is hope. User-developed prototypes can improve communication and significantly improve user satisfaction of delivered systems.

V. The User-developed Prototype: User-developed prototypes can range in sophistication from simple human-machine interface (e.g. menu) simulators to fully functioning prototypes. Regardless of the sophistication, the prototype must consist of four (4) elements to insure the users' requirements have been adequately considered and communicated effectively. The four (4) elements include: functioning hardware and software, a list of necessary data, a brief concept of operations,and written requirements.

VI. Case Studies of HMI and Fully Functional Prototypes: Two prototypes (one of each type) are briefly described. Particular emphasis is given to the circumstances leading to the development of each prototype and the effects each prototype had on the development of operational software.

VII. Impacts: The impacts of using prototypes are significant and worth the effort to improve the efficiency by which limited budgets are converted into operational ADP systems. These impacts include modifying traditional requirements definition/documentation practices and thoughts; loss of total control by the acquisition element of the formal development effort, introduction of competition for the formal delivery by a potentially evolving prototype; and, the need to create, document and distribute guidance to integrate prototypes into today's ADP acquisition environment.

VIII. Advantages: The advantages of using prototypes far outweigh the disadvantages. The advantages include: reduction of the communication errors causing requirements flaws, early validation of requirements, hidden problems discovered, increased user motivation and support, prototyping can be tailored to formal acquisition schedules, substantial cost savings to produce and validate requirements using a prototype rather than the formal delivery, and once developed, the prototype (depending on sophistication) can be used operationally. These advantages benefit not only the users' short and long term needs but also those of the acquisition element and developer. They will have a better idea of the users' concepts and constraints, and thus, can design and deliver ADP products that better meet the users' operational needs.

IX. Disadvantages: The disadvantages of user-developed prototypes include: resistance to change the way ADP has been defined, design and delivered in the past; the potential competition of an evolving prototype versus the ADP system delivered formally; no established procedures to document prototypes and to use that information as requirements; traditional acceptance criteria and test planning must be modified; and, prototypes used for operational work are quick-reaction efforts and, therefore, more risky to develop and maintain over the long term.

X. Recommendation: ESD/XRS (SEIJPO) must initiate a study to determine the feasibility of using user-developed prototypes to legitimately define, refine, validate and document ADP requirements for use in the ADP acquisition lifecycle.

GLOSSARY

1. Automated Data Processing (ADP)--computer software and/or hardware.

2. AFR--Air Force Regulation.

3. Software Support Agent (SSA)--organization responsible for the successful development, acquisition and delivery of ADP systems. Synonymous with AF program offices and performs similar if not identical technical management and contract control functions. Generally separated from the user by location, education and experience.

4. Formal delivery--delivery of ADP system acquired using the formal mechanisms documented in AFR 800-14 and DOD STD 2167 by the SSA developer.

5. Developer--organization responsible for designing and successfully delivering ADP products. Generally private contractors under contract to the SSA but can be government organizations with the assets to design, produce and deliver the needed ADP.

6. Users--organization initiating the request for ADP support. Users include those individuals who will directly use the ADP and also any support personnel to maintain it within the users' organization. For the purposes of this project, "user" and "users" are interchangeable.

7. ASAP--as soon as possible.

8. ACSC--Air Command and Staff College.

9. DOD--Department of Defense.

Chapter One

INTRODUCTION

You are new in the unit and assigned as a user to an antiquated computer system. It is slow and only does a portion of what it should do. Your new boss assigns you to represent the organization and define requirements to upgrade the system as soon as possible. You, of course, dive into the task enthusiastically and make contact with the computer specialists who've been designated to "help you". You find that your organization has only a rough idea of the improvements they want and only minimum documentation to define, validate and fund the program. Ouch! You can't build a system or improve the one you have without something called "detailed requirements". In fact, you discern that these "requirements" are the foundation upon which all future efforts will be based. Yet, you also cannot find a clear idea of what requirements are, or how they should look when completed, or what they should cover to insure your needs are met, or what role you play in the process to translate what ideas you now have to a delivered system. The results: a long agonizing process of soul searching, coordination, frustration and ultimately disatisfaction when the system is finally installed.

Fantasy? Not really. Upgrades to any system can be agony. Computer systems are no exception. In fact, according to a survey from the Oct 1985 American Computer Machine (Sixth AST Notes), 95% of all software delivered does not meet the user's need without some form of modification. (9) What is the problem? Bernard Boar's book, "Application Prototyping," provides a critical clue: within a system's lifecycle (from concept feasibility to system maintenance) 60-80% of "all errors originate in requirements definition". (1:17-18) These errors are caused by the following seven (7) factors:

1. The creation and documentation of requirements is a poorly defined art.

2. The requirements are not usually well understood and well thought out by the users.

3. The users do not exert enough control over the requirements definition and documentation process.

4. Requirements serve many purposes throughout a system's lifecycle which can obscure their original intent.

5. There are "real world" configuration requirements that cannot be documented conveniently and may in fact drive the other requirements.

6. Poor verbal and written communication inhibits the accurate transmission and documentation of ideas from the user to the SSA and developer.

7. There is a long lead time between requirements definition and final delivery and therefore what the user wants now is not what he wants later.

PURPOSE

My purpose, then, is to examine this requirements definition and documentation process and the seven factors causing the errors and recommend improving it through user-developed automated data processing (ADP) prototype systems. These prototypes can and will improve USAF ADP acquisition because they greatly increase the probability of user satisfaction at formal delivery.

OVERVIEW

To make my case, I limited this study by including known assumptions, biases and definitions in Appendix A. Within those limits, I then discuss and evaluate the requirements definition and documentation process focusing on current HQ/USAF/DOD guidance. Next, I model the requirements definition/documentation process as a communication process, describe my experiences as they relate to this communication process and suggest the use of a special "visual aid" (user-developed prototype) to improve it. Then, I define what I mean by a user-produced prototype, why the user must produce it, and describe how it can positively influence the requirements definition process prior to or in concert with the early phases of formal acquisition. I then emphasize my assertions by briefly describing two ADP projects from my personal experience. Finally, I briefly discuss impacts to the current ADP acquisition cycle if this proposal is adopted and the advantages and disadvantages of doing so.

Chapter Two

CURRENT REQUIREMENTS DEFINITION GUIDANCE

What then is a "requirement" ? Who cares? What guidance is available? In order to answer these questions, I'll first define a requirement and discuss why they are CRITICAL to software acquisition. Then, I'll discuss the requirements definition process within the software acquisition lifecycle and evaluate the guidance provided by regulation. I'll close this section with some thoughts concerning the shortfalls noted in the guidance.

REQUIREMENTS DEFINED

Buried in the instructions to fill out AF Form 3125, Communications-Computer System Requirements Document, AFR 700-3 "Information Systems Requirements Processing", contains a typical working-level definition of a requirement:

> [A requirement is] a narrative, expressed in functional terms, telling what needs to be done (not brand names or model numbers). Provide sufficient information to indicate clearly what is to be done rather than why or how. Provide necessary criteria, performance parameters, and assumptions to be used to meet the stated need. Indicate required interface with other communications-computer systems."(6:17)

Although unaware of AFR 700-3, I used this definition for the last four years. Unfortunately, this definition, like the resulting requirements, is too brief, too simple and to sterile to adequately define what is meant by a requirement or how a group of requirements can successfully symbolize user needs. Requirements are a contradiction in that they represent complex concepts fashioned in a complex environment and documented in the simplest fashion possible. But, requirements actually are very complex as you shall see below.

Today, requirements are tersely written statements symbolizing ideas and concepts concerning needs. As defined in AFR 700-3, requirements are intended to be concise unambiguous statements describing all necessary characteristics and functions to design, develop, deliver and maintain a functional system to meet the user's needs. As such, they are produced by committees composed of users, SSA and developers. Requirements range in detail from the very broad contained in Statements of Need and Concepts of Operation to the very detailed contained in hierarchal specifications. Requirements are organized

functionally and heirarchly in documents with titles like "Integrated Fuctional Requirements Document" or "Systems Functional Requirements Document" or "System Specifications" (Note I do not differentiate between requirements and specifications because in fact the only difference is the level of detail). These requirements establish a program baseline and are agreed to by the users, software support agent (SSA) (e.g. the system program office) and the developers.

Requirements definition and documentation is also an ongoing, evolving process throughout the ADP acquisition lifecycle and is the foundation and forum for all activities to successfully define, develop and deliver systems. Any changes are incorporated in an orderly coordinated fashion considering cost and schedule. They are the "center of gravity" in ACSC terms for the successful completion of an ADP project because, through them, the user initiates formal consideration of a need and the SSA and developer understand that need. Requirements are the bottom line against which the users compare the delivery during acceptance testing and are "the standard against which software quality is evaluated". (8: 15).

Requirements are also divided into categories and, as stated earlier, taken as a whole theoretically describe all there is to know about the user's potential system. There are four typical categories: functional, performance, maintainability and reliability. The functional requirements describe what the software must do and what, if any, interaction there will be with the user. For example, functional requirments might be written as "A color Mercator world map projection shall be displayed from any menu upon user request using a single keystroke" or "A color histogram of daily earnings shall be printed". Performance requirements describe the manner in which the functional requirements are executed to meet the users need: for example, how fast, how many ect. Performance requirements may be written separately or integrated into existing functional requirements. For example, the previous two examples can be rewritten to include performance: "A color Mercator world map projection shall be displayed from any menu using a single keystroke within 10 seconds" or "A color histogram of daily earnings shall be printed within 1 minute of final keystroke executing the user's print request". Maintainability and reliability can be lumped under general system requirements and quantitatively describe what they imply. This category is difficult for average users to contribute to and generally falls to the purview of either the SSA, the developer, or the users' computer support branch. For example, a reliability requirement might be worded as "The mean-time-between-failure shall exceed 168 hours" or a maintainability requirement might be worded as "The system shall be capable of daily disk-to-disk backups for all data bases".

MECHANICS OF REQUIREMENTS DEFINITION

Requirements are traditionally developed by two cooperative mechanisms choreographed by the SSA: working groups and interview/inspection of the user's workplace. In the former, the working group (composed of user, SSA and possibly developer) begins with general statements (concepts) describing ADP needs. Each statement regardless of the amount of detail in the beginning is then examined in excruciating detail. This examination yields refinements and clarifications to existing requirements and spawns lower level more detailed requirements. In an iterative fashion, the developer drafts, the SSA supervises and the user reviews each iteration of requirements. As the requirements are "refined", parent requirements spawn daughter requirements etc. Theoretically, as the number of requirements grow they communicate the necessary concepts in sufficient detail for the design process to begin. This process supposedly helps ideas converge between the SSA, user and the developer about what has been asked for. The process continues until the SSA decides the requirements are clear enough for development to begin. My experience has been that no matter how many requirements are written, much is still not understood and the process is really halted when a requirements documents suspense is pending. Each level of documented requirements take on a life of their own and supercede the requirements coming before. This causes much anguish on the part of the user who must constantly verify the requirements have not lost any critical components. The SSA and developer begin this process with the users' concepts and may supplement them using requirements obtained through interviews and study of the user's workplace.

In simplified terms, and according to traditional system analysis procedures, the requirements definition/documentation process involves investigating the users' environment to determine what the user currently does, how he does it, and what data he uses. The system analyst then integrates what the user says he would like to be able to do in the future. The information is gathered by a team of computer experts (presumably) composed of SSA and developer by observing the user's work place and conducting interviews. The intention is to document all jobs or "processes" that take information (data or "input") and produce output. The output might be an individual's action (e.g. put square peg into square hole), and/or textual material, and/or graphical material. The textual material can be words structured in paragraphs (letters, narratives), and/or numerical calculations with tables of numerical output. Graphics

might include color and/or black and white histograms, pie-charts, bar charts and illustrations. This phase is completed (theoretically) when the interviewer is satisfied he has seen everything, documented it and coordinated the results with user to insure comprehensiveness. Unfortunately, the interviewer is converting what he observes in the users environment within the users frame of reference to that of the software engineer in preparation for the next phase--requirements refinement, documentation and specification production. This conversion causes tremendous problems that I'll discuss in the next Chapter.

CURRENT REQUIREMENTS DEFINITION PROCESS

This section is intended to place requirements definition in perspective and provide the background for the evaluation of AF guidance that follows in the next section in this chapter. Therefore, I'll briefly discuss regulations and standards guiding each step of the definition process. At each step of the requirements defintion process I'll evaluate the DOD guidance and assess its contribution to understanding the nature of requirements or defining the roles and responsibilities of the various participants. The requirements definition process flow is depicted below in Figure 1. Note the process begins with the users conept defintion and culminates with formal acquisition. Within the formal acquisition lifecycle, continuing and evolving requirements definition and documentation occures well into Full Scale Production of the delivered product.

As shown in Figure 1, this process is conceptually divided into two overlapping phases separated by levels of detail: concept definition and formal acquisition. The first phase, Concept Definition, begins when the user documents his needs, either informally or for external review and approval. This phase ends upon completion of the user's Operations Concept.

The users' high level needs are formalized in a brief comphrensive statement of needs and is reviewed, validated and, if approved, given funding. For large programs AFR 57-1, Operational Needs, is the guiding regulation. It provides necessary formats and routing. It is strictly used by users to document their needs. In order to complete the Concept Definition phase, further refine user needs, and prove that a concrete plan exists to use the new ADP system, the users will also produce a concept of operations. For large systems under AFR 57-1, AFR 55-24 is mandatory and guides the effort to document the System Operational Concept to "employ, deploy and

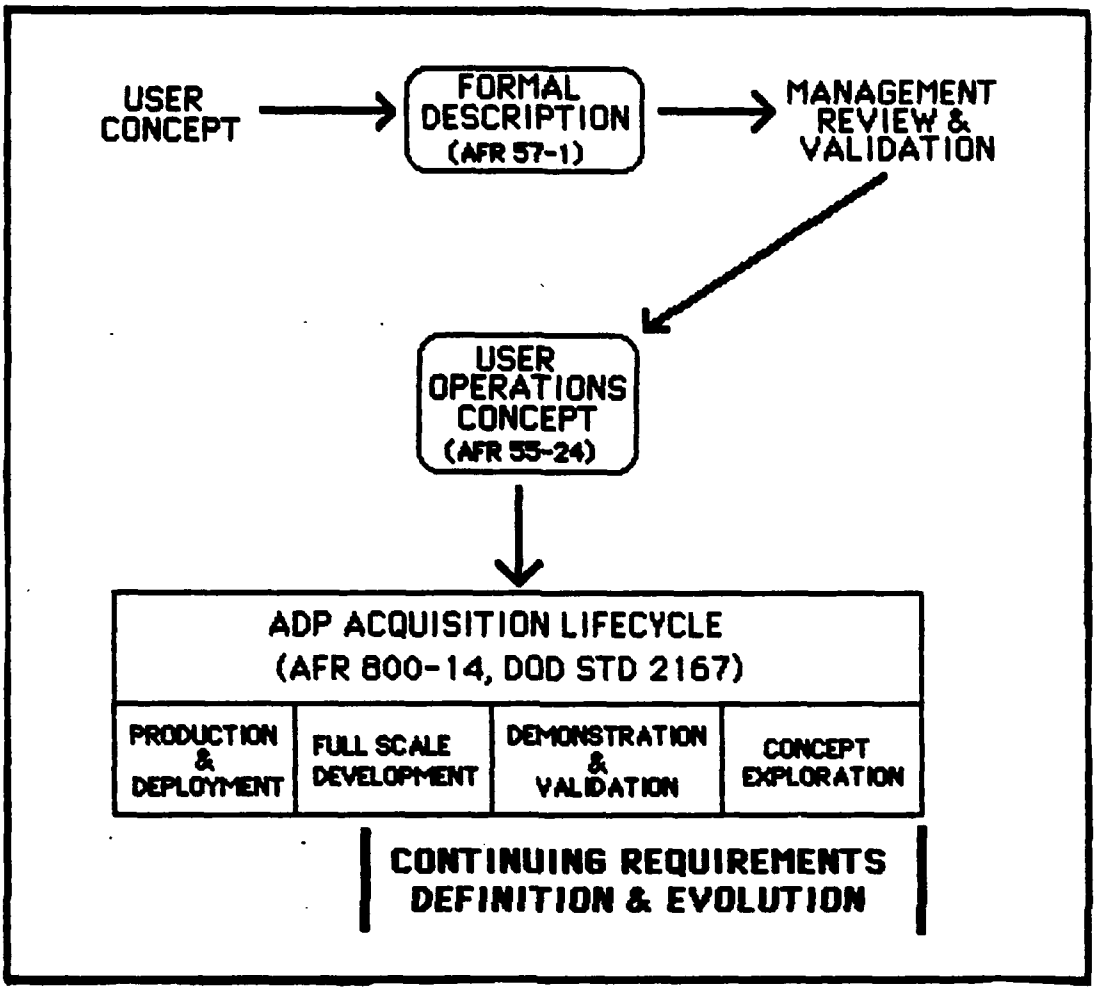


Figure 1. Total ADP Acquisition Lifecycle

support" (4:1) the new ADP system. Although specific timing differs between organizations, this effort is usually concluded prior to the beginning of the formal acquisition or in the case of AFR 55-24 "early in acquisition cycle." (4:1) Like AFR 57-1, AFR 55-24 is a high level document that does not provide much working level guidance to define and refine needs in anticipation of formal acquisition. Tragically, with little guidance, the users will probably spend months if not a year or two in this phase before formal acquisition only to find out some of their notions were and are ill-conceived and not possible!

The next phase is the long term formal acquisition phase. Formal acquisition, or acquisition lifecycle, is regulated by AFR 800-14, "Lifecycle Management of Computer Resources in Systems", and DOD Standard 2167, "Defense System Software Development." The acquisition lifecycle is divided into Concept Exploration, Demonstration and Validation, Full Scale Development, Production and Deployment (or delivery). The first two phases are most critical to the requirements definition and documentation process wherein most of the requirements definition and documentation work is accomplished. Within the Concept Exploration, requirements and operations concepts are analyzed by computer specialists. Requirements inputs are provided by the users as required in working groups (to be covered later in this section). The Demonstration and Validation phase include efforts called "Requirements Definition" to determine "a preliminary allocation of requirements between hardware and software. Document the requirements. . . in a draft Software Requirements Specification (SRS)." (7:13) Without going into detail of each phase, the lifecycle represents an orderly approach to translate requirements into ADP systems with continuous interactions between the users, the SSA and the developer. Now what are their respective roles?

800-14 spells out the following responsiblities for the users during formal acquisition: The users will participate in a working group chaired by the SSA. The users will "provide computer support within system requirements in accordance with AFRs 55-24 and 57-1." (7:2-4,2-5) This infers only top level needs (stated formally by the users) are all that is required for the SSA to begin the project. It clearly shows how limited the users' responsiblities are under current guidance <u>for his own requirements</u>. AFR 55-24 spells out the users' responsibility to "approve concepts for those system development, acquisition, modification or OT&E [Operational Test and Evaluation] not approved by HQ USAF." (4:5) Additionally, the users are responsible for providing a "technically feasible detailed description of the operational characteristics and capabilities of the system required to perform the stated mission." (4:5) This places the burden of ensuring that the high level concepts (i.e. needs) are correctly communicated to (and subsequently acted upon by) the SSA throughout the design cycle on the user. Again, there is no guidance to translate these concepts to detailed requirements or to tell who should do it. What then are the roles of the SSA?

According to AFR 800-14, the SSA is charged with the responsibility to plan, develop, acquire, turnover, and transfer

the assigned ADP system to the user. As such they produce all planning documents (e.g. Program Management Plan (PMP)), selects the acquisition program manager, maintains total lifecycle view of the acquisition effort, and chairs a working group that is the focus for all management efforts related to the planning, acquisition, development and maintenance of the ADP support system. (7:2-1) Within AFR 800-14, I found no specific reference to the responsiblity to define, refine and document users' requirements prior to developer involvement. But, from my own experience, I know the SSA assumes this role as a means to bridge the gap between top-level user concepts discussed in the previous paragraph and the detailed requirements necessary for formal acquisition.

Once the acquisition cycle begins, according to DOD STD 2167, the developer is then responsible for further refining the requirements off-site. The developer is controlled by the SSA with user participation and review. In this phase, the developer uses his experience to draft a hierarchy of requirements describing, in "computerese", each process, input and output identified in the first phase. The SSA and developer use their best judgement to organize the material, get it reviewed by the user, and approved by the user/management (formal document). The goal of this phase is to not only produce accurate requirements reflecting user needs but to segregate them into functional blocks for easy development. To insure quality, the developer establishes and maintains " a complete set of requirements for the software. These requirements shall serve as the standard against which software quality is evaluated" (8:15). Specifically the "contractor [developer] shall define and analyze a complete set of functional, performance, interface and qualification requirements for each CSCI [computer software configuration item]. These requirements shall be derived from the system requirements as defined in the System Segment Specification (SSS), prime item specification, critical item specification, or other sources specified in the contract." (8:19)

EVALUATION OF CURRENT GUIDANCE

These regulations and standard form a hierarchy that takes the user from concept to contract (or a government acquisition office) and therefore more detail is required in each succeeding level. Several conclusions can be drawn:

1. There is not a universal definition of requirements within the regulations and standard nor one that is really adequate.

While, a limited definition was provided by AFR 700-3, it was not contained in any of the other regulations or standards discussed in this paper. Therefore, it is not visible enough. Additionally, the restriction to describe "what" not "how" is intended to preserve maximum flexibility for the SSA to deliver an optimum product in whatever fashion is required to meet the requirements. I disagree with this philosophy and will show in the next chapter that requirements are by their nature inherently confusing and therefore should be constrained in their interpretation by whatever means the user can employ to get his message across.

2. Consistent guidance about the nature of requirements is lacking. To write or contribute to requirements definition, one should have an idea of where one is going and what it looks like when you get there. Specifically, the users, the SSA and the developer have no guidelines when a requirement is done. They simply agree that the words on paper unambiguously represent a users' need. It may not, in fact, communicate the need apart from circumstances in which it was documented. But, the users defer to the SSA and developers because of their their position and training/education as "systems analyst" or "computer scientists or engineers".

3. The regulations and standards reviewed are completely inadequate guides that focus on the SSA and developer roles, neglecting the user who should be the focal point in the requirements definition process. The user is only charged with developing the top-level concepts and needs statements that form the most basic requirements for ADP support. The regulations de-emphasize the user as the process becomes more detailed and emphasize the SSA's responsiblities. The expert in user requirements is the <u>user</u> not the SSA. By this action the user loses initiative and creativity (and some sense of responsibility) and is placed in a position of <u>reacting</u> to someone elses ideas.

4. Because the SSA becomes responsible along with the developer for creating, refining and documenting requirements, the acquisition begins to take on a life of its own. Now it is more important how the software is developed than what is actually delivered to the user. After all, once the acquisition begins, it is the SSA's head if the project is not on time and on budget. There is a great desire to freeze a requirements baseline very early in the development cycle and to consciously or unconciously resist changes because of potential cost and schedule impacts.

Well, if the requirements definition process is unreliable as I hope I have shown in the previous discussion, what is wrong with it and how can it be improved?

Chapter Three

COMMUNICATIONS MODEL OF THE REQUIREMENTS DEFINITION AND DOCUMENTATION PROCESS

The requirements definition and documentation process is a communication process where the requirements are the messages from the sender (or user) to the receiver (or SSA /developer). See Figure 2.

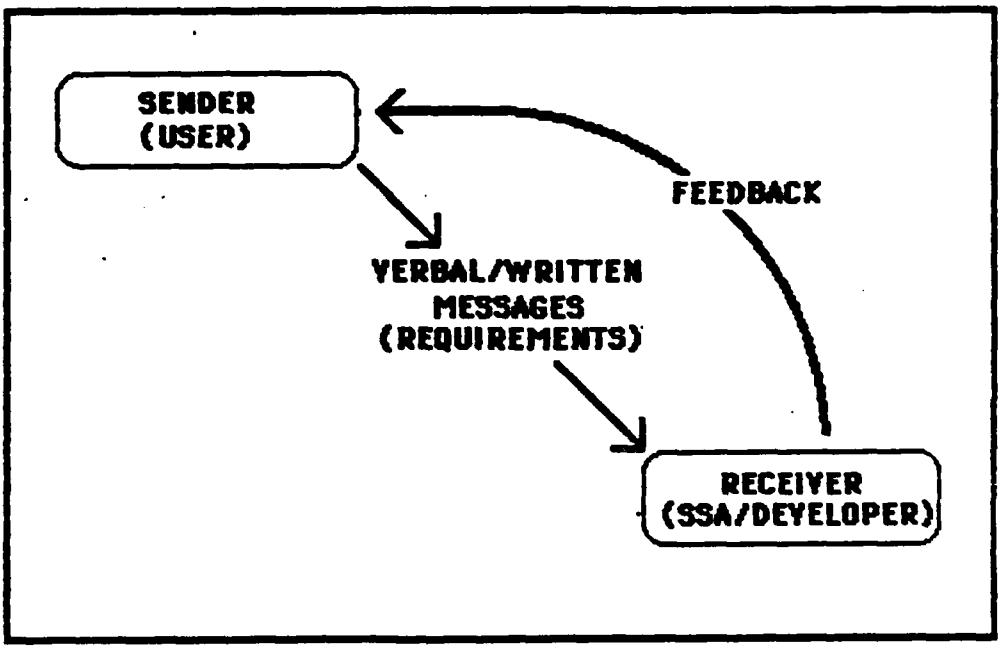


Figure 2. Requirements Communication Model

This requirements communication process involves all senses and includes both verbal and written forms. Verbal

communication is required to extract, transmit and refine the concepts and ideas from the user to the SSA/developer. The written form is used to document and transmit those ideas and concepts in a standardized format, called requirements, to a wide audience for review and later for development and delivery. This communication process is flawed for three basic reasons; currently, requirements are sterile brief statements that do not adequately support the effective communication of complex ideas; the divergent responsiblities and backgrounds participants (SSA, user and developer) prevent the SSA and developer from fully understanding the concepts behind the requirements as a user would; the environment in which the communication process occurs cannot fairly promote communication among the participants. I will cover each of these areas within this chapter and will begin with a discussion of the communication process. As you read the following discussion, bear in mind my previous description of the nature (format and content) of ADP requirments.

EFFECTIVE COMMUNICATION--A DEFINITION

The following excerpts are from two communication experts and describe effective verbal and written communication pertinent to this discussion. Adequately articulating and correctly documenting ADP requirements should follow the same conventions and rules described in these excerpts to be successful. But, that is not the case! Verbal communication, the first part of requirements definition is described in the following way:

> Communication involves a person attempting to use symbols (words and signs) in such a way as to create meaning in the mind of another person. . . . The information that is to be transmitted is called the message. . . effective communication involves all the senses. . . (3:5-6)

The next excerpt not only defines verbal communication but discusses common problems that prevent effective communication. See Figure 3. Shared experience is proportional to understanding. Its importance will be amplified later in this chapter. These problems are dramatically present in the requirements definition process:

> Messages in formal communication are conveyed through symbols. . . words or nonverbal cues that represent . . . actual objects, ideas or experience. . . . Because words and tones and gestures only stand for the real

thing, they are easily misused and misinterpreted. One symbol can mean many things to different people, and what one person chooses to stand for something is quite often not what another person would choose. . . . Receivers do not always get the same message we send. Receivers translate the words we use, and the order in which we use them, into words and orders that make sense for them. (4:6)

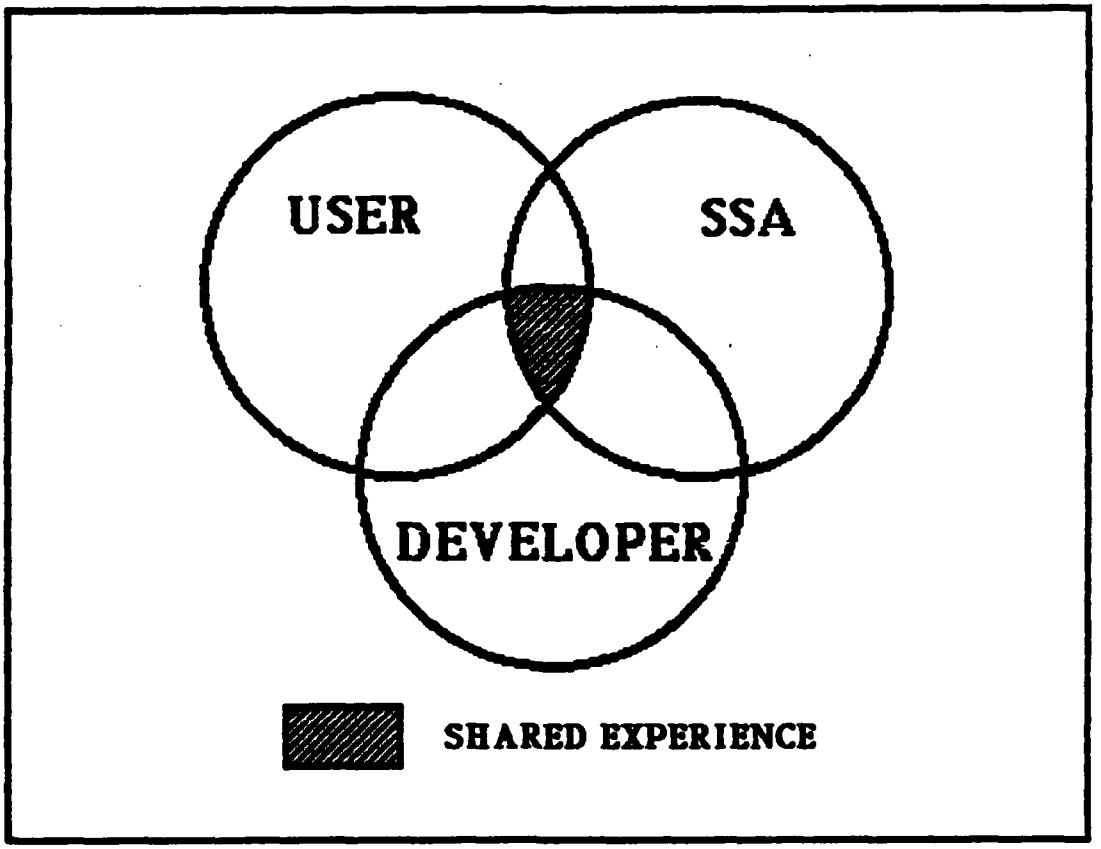


Figure 3. Shared Experience in Communication

The following excerpt focuses on peculiarities of written discussions which are analyzed, sanitized, homogenized and documented into a standardized format of terse numbered narratives.

> Written communication requires precise use of form and mechanics: punctuation marks, pronoun usage, subject-verb agreement, and so on. Since written communication is observed primarily with our eyes, it lack tone of voice and gustural cues. Writers must compensate for these missing components by following semantic (meaning) and syntactic (structural) conventions or rules. Punctuation is the writer's equivalent of the orator's facial gestures and tones of voice. A dash used effectiely can mena "get this." the exclamation point (!) can emphasize irony, warn of disaster, or show exciteament. In other words, punctuation suggests tone. Adjectives, adverbs, phrasing, and clause placement can also create emphasis like that of the gestures or oral communication. . . . Good writers use the semantic, syntactic, and mechanical devices of the language. When you think of a television show, a mental picture quickly flashes in your mind of the actor or actress who represents that show. . . .As a writer, you must show your audience this same kind of visual detail in words. . . . One of the most difficult task for a beginning writer is to create a scene for the reader, a scene that will take what is in the writer's mind and move it to the page and then into the mind of the reader. . . . Formal written communication requires a special and consistent style that will make it acceptable in a number of contexts in which it might be read (4:11).

From the above, you should have an appreciation for the difficulties associated with communicating complex ideas and the ambiguities represented by sterile requirements. The users, the SSA, and the developers conduct the verbal communication as described. Each participant uses gestures, makes faces, refers to examples (i.e.pictures, charts, messages etc) to communicate. The SSA and developer then document what they heard for later review and analysis. In this documentation and review effort, the written requirements are syntactically sterilized. Within each requirement, each word is analyzed and clarified until the requirements do not and cannot communicate effectively without the presence of an original participant.

PARTICIPANTS' RESPONSIBILITIES AND BACKGROUND

The SSA

The SSA was composed of computer literate civilian and/or military personnel. The vast majority had some sort of computer science, computer engineering or information systems undergraduate degree and many have advanced degrees. The SSA had their own language and jargon based on their education and training peculiar to the computer science field. They were very patient, persistent, analytical and precise. In some cases they did not have in either the computer field (engineering field yes--computers no) or previous experience acting as an acquisition agent. In most cases individuals had some operational experience but none directly related to the user. They generally had high confidence in their computer science abilities. After analyzing the users' needs, the SSA began suggesting their own ideas for improvements and interpretations into the requirements. The SSA was extremely concerned about the legal and contractual aspects of any development and expects to control all aspects of the development and acquisition. The SSA began to micromanage the requirements definition process once it had been completed and resisted any changes to the requirements. The rationale for this micromanagement was to create a baseline set of requirements correlated to a fixed acquisition cost. Any changes to requirements therefore required changes to the program baseline.

The Users

The users, on the other hand, were also composed of both civilians, and active duty military. Their formal education was varied and encompased technical schools and/or undergraduate and advanced degrees in various subjects. They had one thing in common; they are all graduates of a specific school of experience necessary to do their operational job. As such, the users, like the SSA, had their own language and jargon based on their operational environment. The users were characteristically impatient, short sighted and fearful. They did not want an extended development cycle; they wanted the software delivered with minimum delay. The users quickly grew bored with anything but operations. They were also fearful that the desired ADP will replace or substantially change the job they do. The users were not sure of all of his requirements or how to articulate them. The users introduced requirements whenever they become clear and

expected them to be incorporated into the next delivery. The users did not understand or like the hierarchal requirements process and quickly got lost as requirements were dissected and combined with other requirements to make a complete requirements document. In fact, after several document reviews and reorganizations, several users actually needed help to locate his own requirements now written in "computerese" ("computerese" are words and phrases commonly understood by computer experts and not normally in the users' lexicon).

The Developer

The developer forms the third leg of this communications triad. The developer was generally a civilian contractor with extensive academic credentials and experience developing computer programs. Contractor personnel generally have advanced degrees (masters and PHD) and many had some sort of military background. The military background and experience did not directly relate to users' operational environment, but enough similarities existed to exasperate problems understanding terminology. Additionally, the developer was insulated from the users by the SSA and/or geography except when formally scheduled. The developer was generally very patient (knowing full well that his time will be paid for whether or not progress is actually made), curious and seeking intellectual challenges. The developer was motivated to deliver a successful product because it enhanced their company's esteem, enhanced their own professional status, and perhaps, most importantly, because it was personally or intellectually satisfying. The developer expected clear unambiguous direction and specification from the SSA and was constantly frustrated by the political battles between the users and the SSA.

REQUIREMENTS DEFINITION/DOCUMENTATION ENVIRONMENT

The requirements definition/documentation process is further complicated because it is conducted in two different and equally unsatisfying environments: the users' workplace and a quasi-neutral meeting room (conference room in users building, SSA or developer) to isolate the user from day-to-day cares. In both cases, the environment reduces effective communication because of irritants or constraints. These irritants and constraints depend on the actual environment in which requirements definition occurs and probably cannot be appreciably reduced because they are constants in the process producing "friction and fog". To illustrate this point, the following describes the environment at my last assignment and is based on observing a select group of users, SSAs and developers in the ADP field (I believe you will see similiarites with your operational environment).

Observing the users' workplace and conducting interviews in-place is critical to understanding the users's job and point of view The users' workplace favoreds the user and his/her attempts to communicate because the user was comfortable, can better control the interaction and there were real samples readily available to supplement interviews and observation periods. Within the workplace, the SSA and developer were intruders who, however well-meaning, were seen as an interruption in day-to-day operations. But, they were tolerated because the users generally understand the need analyze and document the users job(s) in terms the computer scientists (SSA/developer) understand and will facilitate the computer engineering effort (input, outputs, processes, data sources). During this period, the user was subjected to countless questions and redundant conversations that eventually offended the users and reduced their enthusiasm.

Effective communication was reduced because the work-place interviews and observations inherently introduce frustration and irritation for five (5) reasons that negatively biased the participants. First, the users work place did not have adequate seating, space or ventilation to host visitors. Second, day-to-day activities and emergencies distracted the user. Third, interviews were scheduled in advance and oftentimes conflicted with a last minute crises. Fourth, need-to-know prohibited admittance or restricted movement/observation within the users' workplace thereby complicating the data collection effort (this is particularly troublesome if the user has hidden uses for the desired computer support or sensitive data) Fifth, the user did not see the same interviewers all the time thereby increasing the user's frustration with redundant conversations and apparent lack of continuity.

In an attempt to overcome some of these difficulties, requirements definition and documentation continued in a quasi-neutral area. It was intended for the convenience of the SSA/developer and to remove the users from the distractions of their daily jobs. Attending users advocated specific needs and the aggregrate represented the total requirements. The actual meeting place offered advantages not found in the users workplace, such as seating, conference tables and ready access to audiovisual and/or computer support equipment. The requirements were discussed by functional area. Therefore, user and developer attendance varied according to the technical area under discussion. Few developers, users or SSAs had the ambition and persistance to attend the entire phase and some continuity was lost. It was the developers' responsiblity to prepare and present draft requirements based on the first phase analysis of the users' needs and direction from the SSA and to document all user comments. The SSA acted as mediators.

As with the observation/interviews in the users' workplace, there were some problems with this quasi-neutral forum. First, the users were uncomfortable; they did not know what was expected from them or what to expect. Second, they were frustrated at being separated from their job to support an activity that made them feel uncomfortable. Third, the users were frustrated with the process because they could see no immediate support forthcoming and the conversations appeared redundant. This made the users less responsive and helpful. Fourth, the users had a tendency to repeat words and phrases because the SSA and/or the developer did not seem to understand (see the fifth reason). Fifth, because the developers and some members of the SSA attended only specific technical discussions, they had only partial knowledge of the requirements and formed misconceptions about the rest. Sixth, it was difficult to bring the necessary examples to the meeting to clarify requirements without considerable effort and therefore it was not done.

In summary, requirements definition and documentation is a communication process adversely affected by the form of the requirements themselves, the environment in which the requirements definition/documentation takes place and the backgrounds and experiences of the participants. As a result, the communication is biased and the formal requirments in written form cannot effectively communicate the complex ideas symbolized by the requirements. Major effects include:

1. The user, SSA and developer have different education and experience with respect to each other and therefore a different working language and jargon from each other. This reduces the overlap (cross-hatched area of intersecting circles) depicted in Figure 3 and therefore the effectiveness of communication as shared experiences contribute to understanding.

2. The user does not or cannot communicate his ideas effectively because of feelings of confusion, frustration and insecurity about the impact the new system will have on all or part the his job.

3. The user is concerned (and rightly) that the "true statement of needs" will be lost in the communication confusion of the acquisition process.

4. The SSA is generally paranoid about completely controlling all aspects of acquisition in order for it to be successful, particularly the requirements baseline. Any new or

modified requirements are treated with reluctance. Therefore, it behooves all participants to do the best job possible at the outset to define, refine and communicate as many requirements as possible.

5. The SSA and developer strictly adhere to established acquisition philosophy. In other words, it becomes more important how the software is produced, validated and subsequently controlled than it is to necessarily satisfy users' needs.

6. The users are interested in short term answers to solve immediate operational problems and are in direct conflict with the SSA/developer who are generally interested in longer term program implications. The users think in terms of days and weeks, the SSA/developer in terms of months and years.

What can be done to improve this process and increase the chances of satisfying users? The best chance for success is to reform the way requirments are communicated and articulated by the user. How? Previous discussions stated that the user provided samples (e.g. drawings, charts, letter), or visual aids, to clarify ideas and intent. As one communication expert states these visual aids are "one of the best techniques that can be used to facilitate information transmission. . . [and] a visual aid can be anything the audience can see that help get [the] message across." (3:215) Pictures, films, audio aids, videotapes, graphs and full-scale models are all examples of effective visual aids. (3:218) Intellectually, an ADP prototype can be considered a complex visual aid, and therefore it can be used to improve the requirements definition/communication process. As Bernard Boar states "The solution to the communication gap is not to try to make everybody a professional specifier but to permit everybody to receive specifications in a familiar medium. Working prototypes are a common sense means to accomplishing that." (1:34)

Chapter Four

RECOMMENDATION TO IMPROVE THE REQUIREMENTS DEFINITION/DOCUMENTATION PROCESS

The communcation process described in Chapter 3 can be significantly improved through user developed ADP prototypes, in concert with a written concept of operations, a data list and written requirements as necessary. See Figure 4. The prototype and supplementary data will provide access to the "visual" and performance dimensions of the users' concepts which are so difficult to communicate and so difficult to satisfy. In this Chapter I'll answer two questions: What is a prototype? and Why should the user produce it?

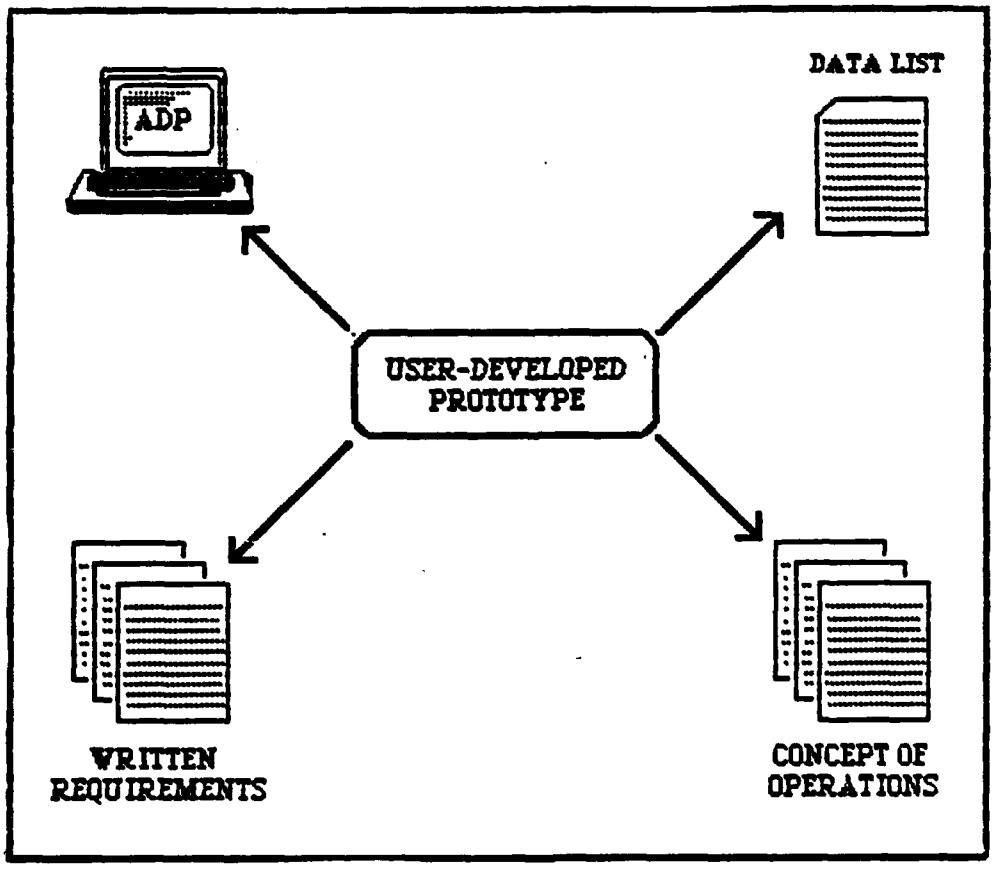


Figure 4. Concept of a User-developed Prototype

WHAT IS A PROTOTYPE?

A user-developed prototype is a low cost, minimum effort, software/hardware computer system designed and built with daily or weekly interaction between the programmer and the user. It can range from a strictly in-house project using whatever computer hardware and programming languages are available to something that requires special hardware and software. The constraints determining where in this range the user falls are time, money and in-house talent. Prototype sophistication can increase as more money, time or programming talent is available. Generally, prototype efforts are envisioned to be several days to months long depending on the constraints mentioned earlier, with most in the later category. The prototype process can begin once the user has developed a general concept and a partial idea of required functionality. The user may furnish drawings, sketches or may react to ideas presented by the programmer. The encounter is kept informal and in the users language. There is a focused goal to make the user happy as rapidly as possible. The crucial aspect of this interchange is that the programmer can then take these ideas and in short order (days to several weeks) can create a program reflecting these ideas for user demonstration, use, review and comment. In that regard, prototyping makes use of two important characteristics: rapid development and flexibility. Depending on the comments offered by the user, final selection of hardware and programming language may change in mid-stream to improve functionality and performance. In this case performance includes computation speed as well as input/output operations to the terminal.

Prototyping should begin with the single most critical aspect of the system and the one most difficult to describe and satisfy, the human-machine interface (HMI). This approach emphasizes how the support looks, act and feels to the user. Once established, the HMI and functionality can evolve iteratively as the user becomes more adept using ADP and has new ideas for that support.

As an example of this flexibility, I supervised a prototype effort that began on an IBM PC/AT, high resolution color monitor and a 30 MByte hard disk. An early constraint rapidly draw and redraw multicolored map projections and latitude/longitude grids based on user selection of the central latitude/longitude of the projection. Hardware was constrained to the IBM PC/AT because

that's what the users were upgrading to. There was some flexibility in the choice of language as the PC/AT is well supported. Initially, Fortran 77 was chosen because existing operational software was in that language and possible long term integration was considered. Unfortunately, we discovered within the first week that Fortran graphics was poorly support and unacceptably slow. After spending 3-4 weeks considering Pascal, C and and commercial graphics support packages, it was decided that Microsoft C was the best choice for several reasons: best performance, graphics support, and we had a C compiler on an operational mainframe should future integration become a possibility. In all cases, demonstrations were run and performance noted. Also, by experimenting we found that modifying the PC/AT hardware to include an Intel 8087 math coprocessor appreciably improved performance again by a factor of five for map redraws. As infered, a prototype would not be complete without the required data identified.

This data, or data list, represents a best guess of the unformatted information required for processing in the prototype. This level of detail demonstrates the user has thought through what the required inputs and outputs are for the needed ADP support. My experience has shown the user generally had a rough idea of the correct data needed and the data sources to do the job. Additionally, the user will also have a strong preference for the format of the output which indicaes needed data. For example, the user may wish the prototype to be a data base and HMI front end tailored for specific retrievals to improve performance. The user then states what data ought to be stored in the database and what combinations of data are required routinely versus ad hoc. My experience has shown the user will know what data is required but may not know what the final form of the output is. Not to worry--iteration and flexibililty is the name of the game. Prior to the actual prototype development and in parallel with data list construction, the users must also define and informally document a concept of operations.

The concept of operations is required at the working level to AUGMENT the prototype/data list to place the required ADP in perspective within the major functions performed by the user. This narrative (with wire diagrams as necessary) communicates the intended use of the ADP support system in operational language. For example, I began a $400K prototype effort based on a coordinated five page concept of operations study for an analytical tool. The concept definition was in work for over a year awaiting go-ahead for prototyping. This study defined the scope of the effort and controversial concepts were then resolved early in the prototype development. The controversy stemmed from

disagreements about whether my office should perform certain functions, what data was needed and for what "real" purpose, and who was going to provide the data. The concept of operation concentrated on how the operator was going to interact with the screen and what the screen looked like as critical functions were performed. While five pages may seem too short, it adequately covered what was necessary and provided a forum for review, coordination and eventually agreement to proceed with the prototype. Even with the prototype, data list and concept of operations, written requirements are still necessary.

Even with these additions, written requirements will still be required to document the formal baseline delivery system and certify delivery of capability. The case I've been presenting shows the user focusing on the creation of the prototype as a living breathing statement of requirements and, oh by the way, the user now has something to use in operations. The "living" prototype should be the focus for the SSA and developer not the user's requirements. The SSA and Developer can examine, analyze and dissect the prototype as necessary to understand the concepts involved. They can observe the user using it and they themselves can be trained to use it. After all this, the SSA and Developer can then document the prototype for their records using requirements, if that is an appropriate term. The user will provide written exceptions to prototype capability (in the form of requirements if required) improve performance, reliability and/or maintainability. These requirements should be integrated into the baseline formed when the SSA and developer analyzed and documented the prototype. At delivery, the developer is responsible to the SSA for successfully delivering according to the requirements and the SSA is responsible to the user for delivering a product that meets or exceeds the prototype capabilities except where noted by exception. This is in essence a menu by menu (or screen by screen or output by output) "fly-off". Differences encountered because the prototype has evolved since the formal program was baselined will be integrated ASAP and in fact as changes occur the SSA and developer get copies.

As stated earlier, prototyping can be a spectrum of complexity depending on money, time and programming talent. At one end of the spectrum is the human-machine interface (HMI) simulator and the other end is a fully functioning prototype that may or may not become integtrated into the operational system.

The HMI simulator prototype only contains the software necessary to display and control the screen dispays (e.g. menus). All required fields are displayed, accompanied by any "help" lines that explain those fields. The input devices

work (keyboard/mouse etc) and it is possible to interact with the program to move from menu to menu and simulate entering data. The intention here is to iteratively determine and validate what data the user needs displayed, how should will look (i.e. color vs black and while and resolution) to the user and where it will be located on the screen. The latter is important to organize and group fields routinely used together to improve user interaction and system responsiveness. Additionally, a menu hierarchy can be established and validated to service routine and special operations. The HMI prototype can be implemented in a fairly simple fashion, taking from days to weeks to accomplish depending on the programmer's skills. A preliminary data list evolves from this effort but has not been validated by the user until and if the prototype becomes functional.

The fully functional prototype begins with the same development efforts and goals of the HMI prototype but the goal here is to validate the HMI, algorithms and data to directly support both a formal acquisition and operations. This is a <u>much</u> more difficult and longer term task than the HMI prototype. Data is loaded automatically or manually via menu and updated as necessary by the program or manually. Since this effort is more extensive than the HMI prototype it can take potentially months to accomplish (remember, however, it would probably take months to document the requirements <u>before</u> any programming could begin). The effort it iterative with the most basic functions implemented first, operational experience /validation next and additional functions added as needed. In this fashion realistic functions supporting operations are implemented and performed in a realistic fashion with realistic performance that exceeds user irritation thresholds. See Figure 5.

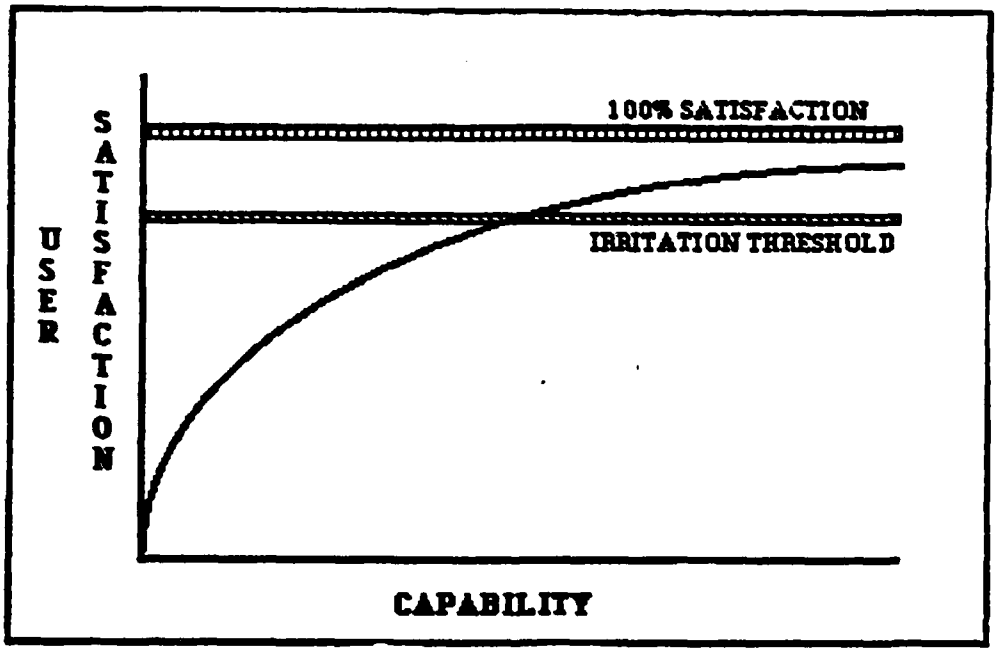


Figure 5. Irritation Threshold

Understanding the concept of user irritation thresholds are critical to achieving user satisfaction. These thresholds are qualitative to the user and quantitative to computer specialists. For example, in user language the software is "too slow," the system is "unresponsive," or the graphics is "irritating." These same categories might be described by a computer specialist in terms of response time in seconds, or search time for a specific number of data elements or a screen resolution of 400 by 400 pixels. The user has no idea what these quantitative thresholds mean but knows what he likes when he sees it and/or experiences it. The curve is qualitative to illustrate that prototyping (and in fact formal development) iteratively seeks to exceed these thresholds until the user is satisfied. The user may raise the thresholds based on new expectations and most probably will never be 100% satisfied. Now, why should the user develop the prototype when the SSA and developer have much greater ADP resources and expertise.

WHY SHOULD THE USER PROTOTYPE?

This section discusses five (5) reasons why the user should develop prototypes. This does not preclude the SSA/Developer from prototyping but their emphasis should focus on implementation alternatives in concert with formal development and delivery of the final ADP support system. The five (5) reasons are:

1. The user is most familiar with his requirements. As stated above, the crucial element in successful prototyping is user-programmer iteration and rapid evolution of the users requirements. The user must and will consider his job, what place the ADP support will have, what the ADP support should do to include input data and desired output, how much machine assistance is reasonable from an operational view. My experience has shown the user will have a general feel for what is reasonable for his environment but needs to be shown other ADP possibilities because his/her view will be limited. For example, the user may not know he could effectively use a color display and such possibilities must be pointed out to him verbally or provided as implementation options.

2. The user is going to iteratively evolve requirements using some sort of forum for review and comment in order to obtain management approval and funding. Regardless of acquisition mode chosen, the user will be forced to conceptualize his needs and communicate those to a second party. In doing so, narratives and pictures are (or should be) generated by the user to communicate his ideas. At this point, efforts diverge: the existing acquisition cycle concetrates on educating the development team (SSA and developer) and producing tersely worded numbered requirements symbolizing the narratives and pictures, or the prototype programmer can begin coding (with some modest

thought to the design) in an attempt to rapidly produce somthing on a computer that represents the requirements. In this fashion, concepts are rapidly translated to code and the project converges on user satisfaction with each iteration until the irritation threshold is exceeded. At this point the user becomes interested other requirements and the process continues. As time progresses, the users requirements may change or evolve by the same process noted here.

3. The user is accepted and has a place to work in the operational environment. This represents less of an intrusion on the user at the most critical phase of development. Additionally, users already understand most of the language and operational concepts and would not need to spend much time in this area to develop a rapore and working knowledge in unfamiliar areas. In my experience, there is a real problem concerning the limited space in the AF workplaces that reduces the effectiveness of any extended inter-personnel exchange and it cannot be assumed away or even solved. Nowadays, it's tougher to get desks and desk space than million of dollars for research and development.

4. Users are now more computer literate because of the personal computer craze. This coupled with improvements in high level languages and hardware capability, and proliferation of computer resources in the Air Force make the possibility of in-house prototyping a reality. I contend the USAF should take advantage of this hobby and allow users with the inclination do prototype projects for their respective work areas to do so. Since much of the work I envision is related to the human-machine interface, it will not take doctorates to write those programs. Simple menus can be as concise as several hundred lines and written by anyone who has programmed a computer game or done any formal programming in high school, tech school or college.

5. The user is not constrained with usual acquisition mindset that worries about formal systems. The primary reason for producing the prototype is to clarify the requirements with a secondary purpose (which I admit comes a very close second) to have something for operations until a formal delivery occurs. With that in mind, prototypers do not have to strictly adhere to formal practices of software engineering like structured programming techniques; formal design reviews; production of user, maintenance and test/evaluation documentation and in-line comments. Each of these elements should be present but the actual implementation may vary from the standards depending on the size of the project, the schedule and the ultimate use for the prototype (e.g. the HMI prototype is straightforward and should not require much documentation, in-line comments or structured programming techniques because it is essentially a throw-away). This lack of formalism increases project flexibility and responsiveness. In prototyping, it is less important to be right the first time (as in the formal development process) than to be close and rapidly converge on an satisfactory solution.

Chapter Five

CASE STUDIES OF HMI AND FULLY FUNCTIONAL PROTOTYPES

This chapter will briefly cover an examples of successful user-developed prototype produced during my last duty assignment: an HMI prototype and a fully functional prototype. Both examples enabled the users to better define, refine, communicate their requirements. Additionally, the production of the prototypes enabled the users to provide unambiguous requirements to an SSA and developer that increased user satisfaction with the next incremental delivery of an expensive operational system.

AN EXAMPLE OF AN HMI PROTOTYPE

The HMI prototype simulated selected menus from the next incremental delivery of the operational system. The information was obtain from the Critical Design Review (CDR) documentation provided by the developer and SSA. As was the normal case, CDR documentation consisted of several hundred pages of flow diagrams and proposed menus that arrived ten days prior to the formal CDR review held at the developer's facility. The users, concerned with only specific portions directly supporting their functions, valiently tried to review what they thought were the critical areas. One group, had significant trouble getting users to review and comment on the material even though it contained the proposed menu hierarchy and functionality for a data base critical to their evolving operations. The users were overwhelmed by the sheer volume of the information, its engineering format, and frustration over the long delay between the statement of requirements and delivery (over two years for a $13M upgrade to an existing system). In an effort to improve this situation, one user, who was an extremely proficient programmer (hereafter called the user-programmer), created a menu simulator and in the process successfully involved 2-3 additional users in its design.

This menu simulator was accomplished in 2-3 days and implemented on an IBM PC using Turbo Pascal. This simple program was several hundred lines long and contained a data base

containing menu layouts forming a hierarcheal tree and a small driver program. There were approximately 10 menus and selected keys were programmed for special functions (e.g. display help information associated with each field) to simulate the exisiting system. The users could tab to each field, enter data and transition between menus in a realistic fashion. There were no connections to any data base. Performamce and responsiveness were based on the characteristics of the stand-alone IBM PC, and in retrospect did not really simulate the multi-user system that was operational. The simulator's performance and responsivenss did, however, exceed the user's irritation threshold and thereby provided a standard to judge future deliveries. Once complete, the simulator was reviewed and used by 2-3 users and modified to include their needs. I would estimate 20-30% was changed in this review. These changes ranged from new or modified fields to reorganizations of menus. This program was then taken to Critical Design Review and demonstrated to the developers and SSA (program office).

This effort accomplished four things. First, it provided an extremely effective forum to create and maintain user involvement. It was real and the user could touch it. Second, the users became intimately involved in the design of "their" data base and HMI and pride of authorship stimulated further interest in the formal delivery but also in the prospects of further prototyping. Third, changes to the data fields and menu layouts were incorporated by the Developer to the extent possible based on limitations of the existing operational code. This was certainly the intention of the user, but the SSA was not enthusiastic about last minute (in their eyes) changes. Fourth, the users were able to more fully visualize the the capability of future delivery and its place in operations, and thus began the arduous transition from a purely manual operation to one with integrated ADP support.

AN EXAMPLE OF A FULLY FUNCTIONING PROTOTYPE

Our first fully functioning prototype was based on the data base needs discussed above for the HMI prototype. After the Critical Design Review demonstration, the decision was made to develop a functional data base and accompanying HMI for three reasons: First, it was obvious that although the SSA/developer incorporated many of the desired modifications, the delivery would not satisfy all the users evolving requirements (particularly performance and hardcopy reports), and after waiting two years, this was unacceptable; Second, the users needed the data base now in order to streamline operations and transition from a manual operation and; third, the users felt they could prototype the data base in a shorter time, satisfy more requirements, and do a better job.

The data base and HMI was divided into two major blocks: a stand-alone IBM PC/XT (with one 5 Mbyte hard disk) version and a VAX 11/780 version using the IBM PC/XT as the user interface. The first block was produced for a the stand-alone IBM PC, tailored to the needs of a single user acting as the data base administator. The users had already prepared and coordinated informal documentation describing 60-70 data fields, the HMI and the desired output format which drove the nature of the retrieval capability. This step was partially validated by the HMI simulator discussed in the previous section. Working closely with the users daily, the user-programmer delivered this block in only took six weeks! It was written in Pascal and was crude, but it did function. Upon delivery, the user/data base administrator began storing data and generating reports for operational use; thereby validating the design and changing poorly implemented functions or eliminating bad ideas. The programmer was able to correct many deficiencies within hours or days and the iterative process continued. Due to severe performance limitations of the PC version, the next six months were spent creating the second and much improved block.

This second and final block was composed of three parts; the HMI on the IBM PC/XT, the data base residing on the VAX 11/780 mainframe, and a tailored communications protocol connecting the PC to the VAX via RS-232 serial interface. The HMI was developed in Turbo Pascal, the communications protocol in a combination of Pascal, machine language and "C" language and the data base in "C" language. A second full-time programmer was added to develop the data base on the VAX. The data base was sized according to expectations (350 requirement records). Throughout the period, weekly interactions took place between the user and the programmers. This block was delivered six (6) months later and contained approximately 4000 lines of code, 3000 in the PC and the remaining 1000 on the VAX (better than industry standard of 100 lines of code per week). As with the Block 1 version, this was a single user system that never evolved beyond that, but operations were adequately supported. The discussions up to this point accent the positive, but there were negatives to be mentioned. First, there was little time to conduct any design review process and so the prototype was a surprise to some users. Second, the effort was tied to the imagination and abilities of only two programmers. Third, maintenance of the prototype once delivered fell to their shoulders because there was little documentation aside from in-line comments. Fourth, there were a number of errors present at delivery that probably wouldn't have been under a formal delivery that tied up the programmers when they could have been producing documentation.

What did this accomplish? First, it produced ADP support that was the mainstay of an evolving organization when formal mechanisms proved unresponsive. Second, it showed that a low overhead prototype project was feasible and could be used to improve operations. Third, this prototype became the user's standard against which the upcoming formal delivery was measured rather than the written requirements; the user was not satisfied if the formal delivery could not give equivalent performance/responsivness and increased functonality. The user was motivated to improve this formal delivery because it ultimately offered greater potential for rapid improvement in future. Fourth, the user had an in-house training and demonstration resource to educate new personnel and provide demonstrations. Fifth and finally, this prototype validated a basic prototyping premise; prototyping can be productive using whatever resources are at hand in today's ADP rich environment.

Chapter Six

IMPACTS, ADVANTAGES AND DISADVANTAGES OF USER-DEVELOPED PROTOTYPES

The preceeding chapter described two examples of successful prototype projects implemented by frustrated users to clarify and evolved requirements. In the case of the fully functioning prototype, (i.e. the data base) it also was used operationally while awaiting delivery of the formal system. Such use could and should be proliferated throughout the Department of Defense's ADP acquisition community to increase cost effectiveness. This chapter discusses the consequences of incorporating prototypes into the formal ADP acquisition cycle. As such, the impacts, advantages and disadvantages of integrating the user-developed prototype, data-list, and concept of operations in connection with documented requirements are enumerated below:

IMPACTS

1. Traditional requirements analysis must be rethought. Prototypes symbolizing requirements, with the force of written requirements, is a new concept and perhaps not easily accepted.

2. The SSA will lose total control of the ADP support. The SSA will only be responsible to improve and proliferate specific capabilities developed and validated by the user's prototype and accompanying documentation (e.g. Concept of Operations).

3. Clear consistent guidance must be developed and distributed so users, SSA and developer can effectively integrate this technique into the existing ADP lifecycle. The prototype process can begin anytime after the original high level concept is drafted to positively effect requirements communication and eventual delivery as a formal system. Therefore, regulations like AFR 55-24, AFR 57-4 and AFR 800-14 will have to be modified to direct attention to the creation of a user-developed prototype and the subsequent analysis/documentation of it by the SSA/developer. This analysis should be the main source of requirements supplemented as necessary via the requirements definition process described in Chapter 2.

4. The users are forced very early in the concept definition process to comphrensively consider all concepts and validate, through operational review and/or use, the requirements.

5. A clear unambiguous statement of requirements will be available in the prototype if it is developed.

ADVANTAGES

The advantages of the user-developed prototype, data-list, and concept of operations in connection with documented requirements far outweigh the disadvantages. These advantages are enumerated below:

1. The users are provided a forum that dramatically articulates and validates requirements. Useless functionality or improper implementations become very evident and are quickly eliminated or fixed.

2. Hidden problems that are inherent in the operational environment can/will be identified and solutions provided before a costly formal development. These problems may or may not impose technical constraints on future ADP delilveries but they add to the uncertainty and confusion of requirements. These problems may be organizational (e.g. Who should be doing this job? Where will the user be located physically and is the user dependent on the actions of another user? Is this a valid mission?) or physical constraints (e.g. How much space is available? Is there sufficient ventilation? How close are the users normally seated?). A good example of this comes from my last work environment: There were 100 users in a long narrow bay that was three or four desks across. Each desk had a bookcase and one IBM PC for every two desks for stand-alone word processing and connections to VAX mainframes. In the course of daily business, the user may move from stand-alone word processing to a message processor to the VAXes and to one of the few special use terminals connected to remote systems servicing our data and computational needs. The process of logging in and out of the mainframes (and their slow response times) and the need to reboot stand-alone programs was very inefficient. This situation was compounded by the lack of space which prohibited lots of terminals on each desk. And yet, the leading solution for several years oscillated between multiple terminals on individual desks and consolidating all stand-alone word processing and message generation functions with the local operational programs and computational programs and data bases on remote hosts. Ridiculous! There wasn't the space, the money or the time.

3. The users' enthusiasm (and subsequent support) increases the closer to delivery they get. The short schedule of the prototype and pride of "authorship" (by personalizing the prototype) peaks user enthusiasm early and sustains it throughout the development of both the prototype and the formal system.

4. The user iteratively describes and validates data needs. There is a danger that without such validation, functions (computation and report generation) and data may be introduced as future capability that are never used. Unused ADP modules are "white elephants" because they are treated as critical and yet their existence depends on somebody, someday possibly using it. For example, we had the capability to produce several hardcopy reports delivered in 1983 to support our Performance Evaluation Directorate. The intention was to provide a rudimentary capability which would mature as the users decided what to do next and as the data became available. Neither happened, and the capability was never used operationally during the four years I was there.

5. It is very difficult to express users' needs for certain elements in written words because they are more qualitative than quantitative. Users' cannot assign a number to describe them but certainly knows what is acceptable. These "irritation thresholds" can only be determined by experimentation/demonstration. The prototype offers this forum. Those elements in this category include and are not limited to performance (responsiveness), CRT resolution (depends on the application and user environment), screen layouts, physical configuration and location of terminals. In my experience, each of these categories require an iterative solution in the users' environment under realistic operating conditions.

6. Prototyping can be tailored to formal delivery schedule and is only constrained by imagination, talent, time and money/ADP. Prototypes can range from a partial HMI simulator to a fully functional prototype. Regardless of prototype sophistication, requirements can be validated better using this technique than if only written requirements are reviewed and acted upon by the SSA and developer.

7. Prototypes are a much more cost effective technique to validate concepts and/or requirements than waiting for the operational use of the formal system. For example, we had a very difficult application project that could not be adequately defined because new functions were to be added to our organization requiring ADP support. The SSA felt the application

tool (if we could define and agree what we needed) would cost approximately $5M and would take two years to develop. We felt very uneasy committing that kind of money and time and began an low-level effort using approximately three contractors for 9 months at a cost of $450,000. In the six months I was associated with the project before my PCS, we made significant progress towards validating our concept, obtaining required data and quelling political problems about performing those functions. The effort continues today and is evolving towards an operational capability that I predict will cost less than the original $5M with much better results.

8. Prototyping creates a more discriminating user and a sense of healthy "computer product competition" that improves the quality of the formal delivery by the SSA/developer. The user already has a benchmark (prototype) with which to compare the new formal delivery. Upon comparing the two, it is conceivable the user won't use the formal delivery and this places pressure on the SSA/developer to deliver quality. This benchmark concept is is synomous with the case of providing an upgrade to an existing system; the user will compare the new delivery to existing capabilities and absolutely will not accept breakage or reduced performance in the name of progress.

9. Prototyping offers a mechanism to correct the communication errors that flaw ADP requirements by supplementing the traditional requirements definition and documentation process described in Chapters 2 and 3. This would be much simpler to implement than to retrain the SSA or Developers, or move into new operational buildings with spacious meeting rooms or address the conflicts between the SSA, developer and users because of their different backgrounds.

DISADVANTAGES

The disadvantages of the user-developed prototype, data-list, and concept of operations in conjunction with documented requirements are enumerated below:

1. The primary disadvantage of incorporating this recommendation stems from uncertainty. Traditional ADP development has been done using traditional requirements definition (as described in Chapter 2) for a long time. There will be significant resistance to "local" efforts because they will seem amateurish and are not supportable over the long term: non-standard hardware is not easily integrated into a total "system design", software is not written and documented according to formal standards etc.

2. The SSA will see the prototype process as a competitor that could put them out of a job. This happened at my last duty station and the result was extreme resistance on the part of the SSA to accept this technique as viable and complementary to their own activities.

3. Depending on the nature of the prototype, users will possibly use and evolve the prototype in parallel with the formal system development. This is certainly has been the tendency in my experience. But, I have observed that users will readily switch when provided a <u>suitable alternative</u> (regardless of source) that allows the users to do his job faster and/or easier. It's up to the SSA/developer to find solutions that attract the users away from the prototype.

4. There are no established procedures to document prototypes and store that information as one would the user requirements discussed in Chapter Two.

5. Since the users measure the formal delivery against the prototype benchmark, any acceptance test planning must be restructured for comparison against the prototype and not written requirements.

6. There may be a proliferation of undocumented unsupport ADP performing critical AF operations if long term maintenance is not provided.

Chapter Seven

CONCLUSION

SUMMARY

In summary, the traditional requirments definition and documentation process used to obtain computer systems is flawed. These flaws reduce the effective communication of users' requirements to the SSA/developer delivering the system. The basic reasons behind the communication difficulties include: lack of adequate and consistent requirements definition and documentation guidance; the nature and mechanics of documenting requirements; the different roles, backgrounds and experiences of the users, the SSA and the developers and; the environment in which the users, SSA and developers must work to define and refine requirements. But, there is hope. The use of user-developed prototypes can improve the communication and significantly improve user satisfaction of delivered systems.

The user-developed prototype, as described in Chapter Four, can range in complexity from a simple HMI simulator to a fully functioning prototype depending on users' resources of time, talent and money/ADP. Throughout the spectrum, the prototype must consist of four (4) elements to insure the users'requirements have been adequately considered and most effectively communicated to the SSA/developer. Those four (4) elements include: functioning hardware and software, a list of necessary data, a brief concept of operations and written requirements. The user-developed prototype will provide a new focus and forum for the user to express requirements and for the SSA/developer to analyze and improve in a formal delivery.

The impacts of using user-developed prototype systems are significant and worth the effort to improve the efficiency by which limited budgets are converted into operational ADP systems. These impacts include modifying traditional requirements analysis/definition practices and thoughts, loss of total control

by the SSA of ADP development efforts, introduction of competition for the formal delivery by a potentially evolving prototype, the need to create, document and distribute guidance via regulation, conference and standards for consistent practices that integrate the prototype into today's ADP acquisition environment.

The advantages of using a prototype far outweigh the disavantages summarized in the next paragraph. The advantages include: reduction of the communication errors causing requirments flaws, early validation of requirements, hidden problem discovered, user motivation and subsequent support is increased, prototyping can be tailored to formal acquisition schedules, there is substantial cost savings to produce and validate requirements using a prototype than the formal delivery, and once developed, the prototype (depending on complexity) can be used operationally and for training. These advantages benefit not only the users short and long term needs but also the SSA and Developer. The SSA and Developer will have a better idea of the users' concepts and constraints. They are freed from worrying about the form and content of the users' requirements and can do what computer specialists are trained to do: design and deliver ADP products that meet the users' operational needs. Thus, the use of a prototype use makes everyone's job easier. But, there are some draw-backs.

On the negative side, the disadvantages of user-developed prototypes included: resistance to change the way ADP has been defined, design and delivered in the past, the potential competition an evolving prototype against the system delivered in the formal delivery, no established procedures to document prototypes and use that information as requirements, traditional acceptance criteria and test planning must be modified, and prototypes used for operational work are quick-reaction efforts and, therefore, more risky to develop and maintain over the long term.

FINAL RECOMMENDATION

I strongly recommend AFSC/ESD initiate a study to determine the feasibility of using of user-developed prototypes accompanied by a concept of operations, data-list and written requirements (as necessary to supplement the prototype) to legitmately define, refine, validate and document ADP requirements.

THE END

BIBLIOGRAPHY

A. REFERENCES CITED

Books

1. Boar, Bernard H., Application Prototyping. New York, NY: John Wiley & Sons, Inc., 1984.

2. Hunt, Gary T., Effective Communication. Englewood Cliffs. New Jersey: Prentice-Hall, Inc., 1985.

3. Katula, Richard A., et al., Communication: Writing and Speaking. Boston, Mass., Little, Brown and Company, 1983.

Official Documents

4. Air Force Regulation 55-24, "System Operational Concepts," 28 February 1986.

5. Air Force Regulation 57-1, "Operational Needs," 28 May 1985.

6. Air Force Regulation 700-3, "Information Systems Requirements Processing," 30 November 1984.

7. Air Force Regulation 800-14, "Lifecycle Management of Computer Resources in Systems," 29 September 1986.

8. Department of Defense Standard (DOD-STD) 2167, "Defense System Software Development," 4 June 1985.

Other Sources

9. Prange, John. PHD in Mathematics. Department of Defense. Fort Meade, Maryland. Telecon, November 1987.

B. RELATED SOURCES

Case, Albert F., Jr. Information Systems Development. Englewood Cliffs, New Jersey: Prentice-Hall, 1986.

Hart, Anna. Knowledge Acquisition for Expert Systems. New York: McGraw-Hill Book Company, 1986.

APPENDIX

ASSUMPTIONS AND BIASES

1. This is not a comprehensive study due to the limited nature of the Air Command and Staff College research program. The regulations and standard used were chosen as typical examples governing ADP acquisition. The regulation or standard itself was not the important issue but rather its treatment of the nature of user requirements and the responsibilities of the participants.

2. The scope of my experience with formal ADP acquisition is restricted to only the last four years. Generalizations are made based on personal observations, impressions and conversations concerning other ADP projects and AF/DOD program offices.

3. My recent experience with ADP was not a positive one for the following reasons:

a. Unfriendly competition and relations with the software acquisition agent who did a poor job delivering ADP support to my organization. That same agent did even a poorer job providing mechanisms to correct the deficiencies.

b. Evolving user mission and therefore large segments of requirements were unknown and/or defied quantitative description and still have operational meaning.

4. I've selected regulations and standards normally used for major programs with the DOD and Air Force. In this regard, I am looking for source documents to verify the existence of consistent guidance for the requirements definition and documentation process. Local policies and regulations were not considered because of the limited scope of this effort and the desire for consistency.

5. I assumed the concepts documented in AFRs 800-14, 700-13 and DOD Standard 2167 are consistently applied throughout the Air Force regardless of the size of the ADP system sought.

END

DATE

FILMED

8-88

DTIC